VANU INC

JTRS Portability Workshop
29 April 2004

Radio Description Language

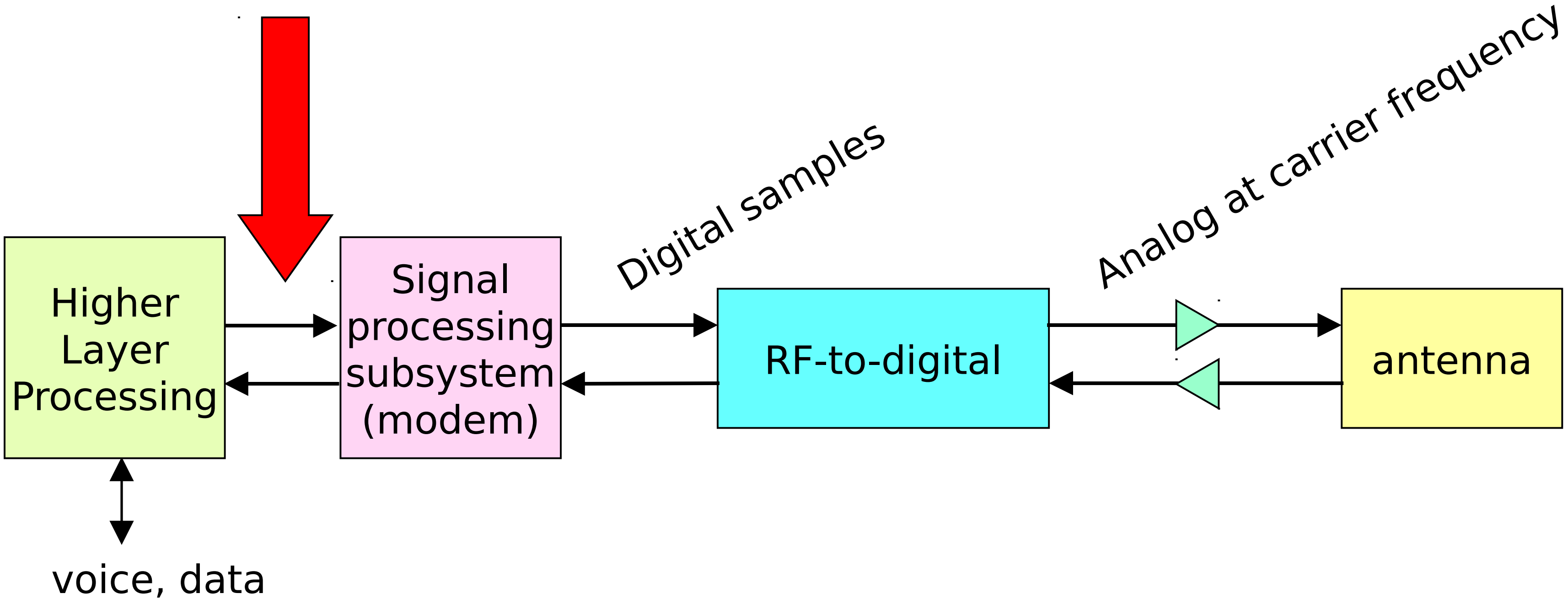
Dr. John M Chapin
Vanu, Inc.
1 Porter Square Suite 18, Cambridge MA 02140
jchapin@vanu.com
617-864-1711

RDL project

- RDL developed 2000-2002 to support multichannel basestations
  - Used for Vanu, Inc. GSM basestation
- Supported 2002-2004 by JTRS JPO
  - Included beta test by external users

Radio Description Language

- An evolution of the SCA modem API
- Improves this interface to make higher layer code more portable



Radio Description Language

- Vanu, Inc. RDL is a language for
  - describing the desired signal processing functions
  - giving parameters for each processing stage

- Modem API shrinks to 1 function:

```
loadRDL(RdlDescription d);
```

- An RDL description
  - is used to configure and control a flexible modem
  - is NOT a language for implementing signal processing
  - is NOT a waveform specification

A language is better than a functional API

1. Reduce code size for large-scale systems

   1000s of things to control in the signal processing subsystem

2. Simplify code for dynamic, complex waveforms

   1000s of modes, dynamically changing processing

3. Improve portability across different SPS hardware types

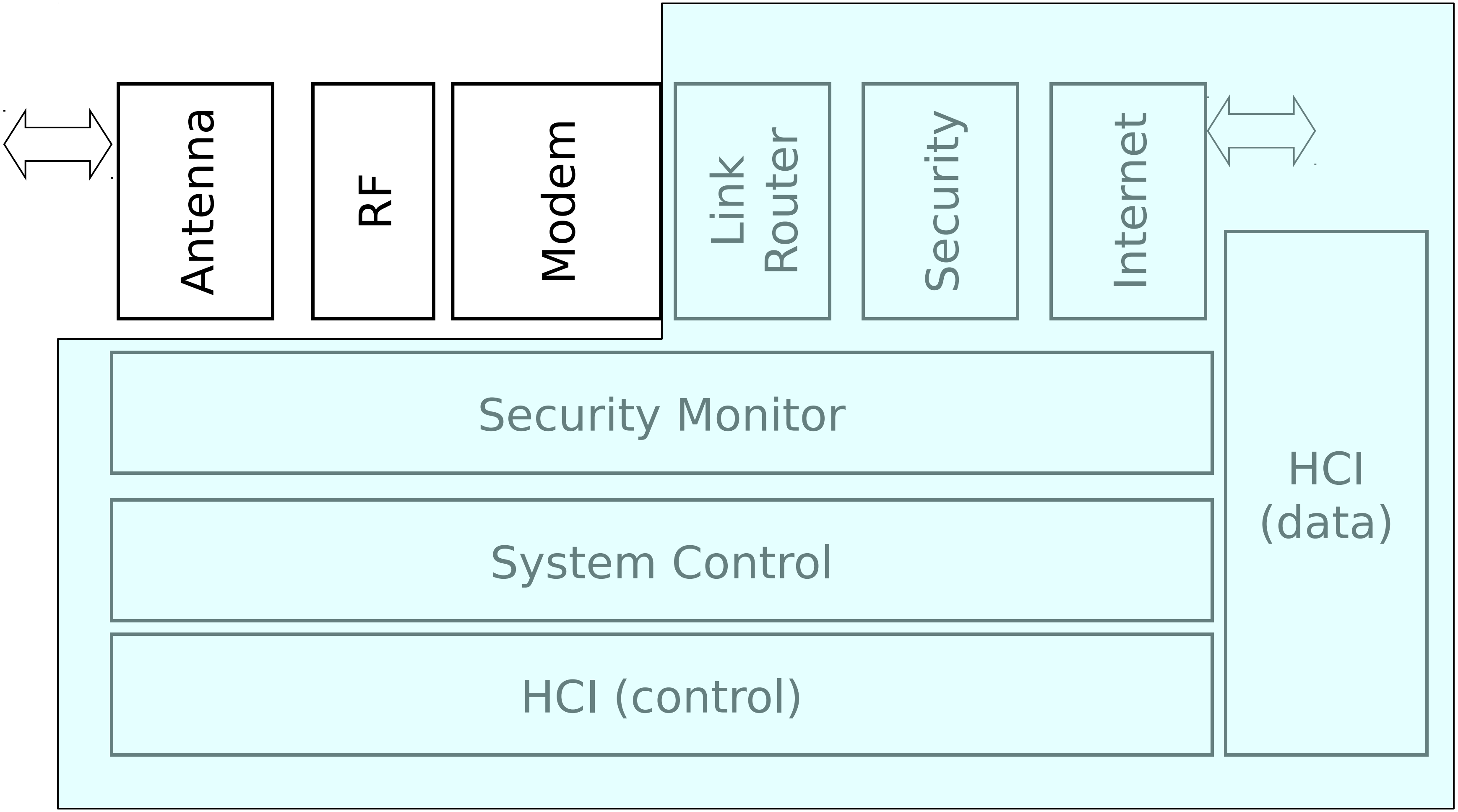
   host code operations change significantly with type

Other alternatives to functional API

- XML
  - XML generator needed for repetition and hierarchy
  - RDL expresses this directly
- Objects on the GPP
  - a model of the signal processing graph
  - can use CORBA interconnect
  - potential challenge for dynamic graph changes
  - higher memory footprint
- Either XML or Objects could be effective

Backup Slides

SCA software reference model



“Higher layer code”

Antenna

RF

Modem

Link Router

Security

Internet

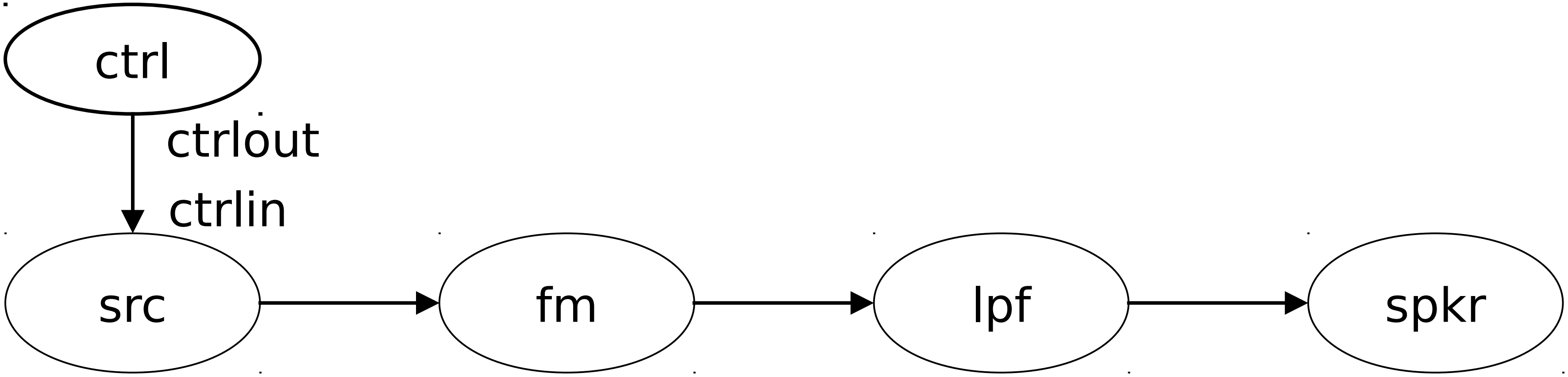
Security Monitor

System Control

HCI (control)

HCI (data)

Host waveform code to configure SPS for FM



ctrl

ctrlout

ctrlin

src

fm

lpf

spkr

```
srcconfig x;
x.freq = 91.50;
x.width = 0.2;
x.multi = FALSE;
x.rate = 500;
config_src(&x);
if (x.error) {
 /* report */
}
```

```
fmconfig f;
f.width = 0.1;
f.sigmax = 256;
config_fm(&f);
if (f.error) {
 /* report */
}
```

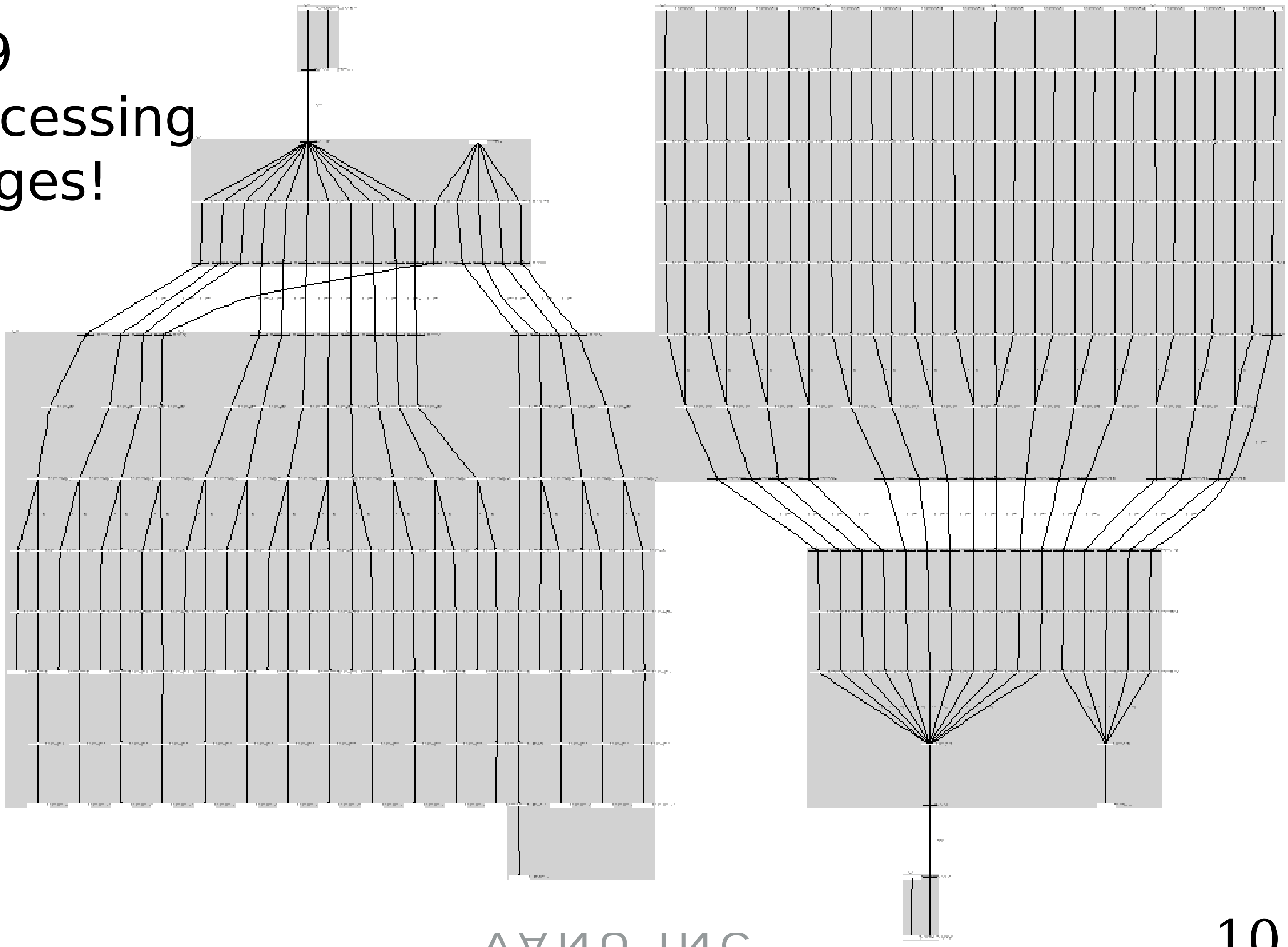
```
lpfconfig l;
l.passband = 32;
l.stopband = 36;
l.passtol = 0.2;
l.stoptol = 0.1;
config_lpf(&l);
if (l.error) {
 /* report */
}
```

```
spkrconfig s;
s.volume = 5;
config_spkr(&s)
if (s.error) {
 /* report */
}
```

(code shown is notional)

Sigproc for 16-channel AMPS basestation

469 processing stages!



Configuring SPS for AMPS basestation

```
l.passband = 32;
l.stopband = 36;
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 802.1;
config_lpf(300,&l);
if (l.error) {
 /* report */
}
l.passband = 32;
l.stopband = 36;
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 802.6;
config_lpf(&l);
if (l.error) {
 /* report */
}
l.passband = 32;
l.stopband = 36;
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 803.1;
config_lpf(&l);
if (l.error) {
 /* report */
}
l.passband = 32;
l.stopband = 36;
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 803.6;
config_lpf(&l);
if (l.error) {
 /* report */
}
l.passband = 32;
l.stopband = 36;
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 804.1;
config_lpf(&l);
if (l.error) {
 /* report */
}
```

```
if (l.error) {
 /* report */
}
l.passband = 32;
l.stopband = 36;
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 804.6;
config_lpf(300,&l);
if (l.error) {
 /* report */
}
l.passband = 32;
l.stopband = 36;
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 805.1;
config_lpf(&l);
if (l.error) {
 /* report */
}
l.passband = 32;
l.stopband = 36;
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 805.6;
config_lpf(&l);
if (l.error) {
 /* report */
}
l.passband = 32;
l.stopband = 36;
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 806.1;
config_lpf(&l);
if (l.error) {
 /* report */
}
l.passband = 32;
l.stopband = 36;
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 806.6;
config_lpf(&l);
```

```
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 807.1;
config_lpf(&l);
if (l.error) {
 /* report */
}
l.passband = 32;
l.stopband = 36;
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 807.6;
config_lpf(300,&l);
if (l.error) {
 /* report */
}
l.passband = 32;
l.stopband = 36;
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 808.1;
config_lpf(&l);
if (l.error) {
 /* report */
}
l.passband = 32;
l.stopband = 36;
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 808.6;
config_lpf(&l);
if (l.error) {
 /* report */
}
l.passband = 32;
l.stopband = 36;
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 809.1;
config_lpf(&l);
if (l.error) {
 /* report */
}
l.passband = 32;
l.stopband = 36;
```

```
l.passband = 32;
l.stopband = 36;
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 802.1;
config_lpf(300,&l);
if (l.error) {
 /* report */
}
l.passband = 32;
l.stopband = 36;
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 802.6;
config_lpf(&l);
if (l.error) {
 /* report */
}
l.passband = 32;
l.stopband = 36;
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 803.1;
config_lpf(&l);
if (l.error) {
 /* report */
}
l.passband = 32;
l.stopband = 36;
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 803.6;
config_lpf(&l);
if (l.error) {
 /* report */
}
l.passband = 32;
l.stopband = 36;
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 804.1;
config_lpf(&l);
if (l.error) {
 /* report */
}
```

```
if (l.error) {
 /* report */
}
l.passband = 32;
l.stopband = 36;
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 802.1;
config_lpf(300,&l);
if (l.error) {
 /* report */
}
l.passband = 32;
l.stopband = 36;
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 802.6;
config_lpf(&l);
if (l.error) {
 /* report */
}
l.passband = 32;
l.stopband = 36;
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 803.1;
config_lpf(&l);
if (l.error) {
 /* report */
}
l.passband = 32;
l.stopband = 36;
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 803.6;
config_lpf(&l);
if (l.error) {
 /* report */
}
l.passband = 32;
l.stopband = 36;
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 804.1;
config_lpf(&l);
```

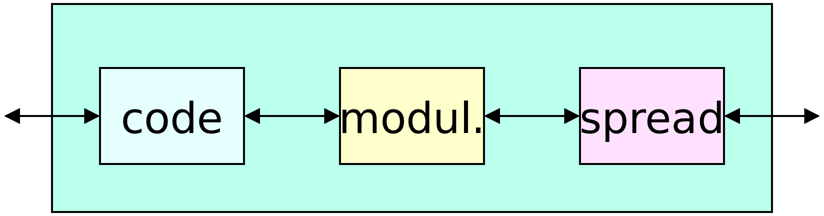
```
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 804.1;
config_lpf(&l);
if (l.error) {
 /* report */
}
l.passband = 32;
l.stopband = 36;
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 802.1;
config_lpf(300,&l);
if (l.error) {
 /* report */
}
l.passband = 32;
l.stopband = 36;
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 802.6;
config_lpf(&l);
if (l.error) {
 /* report */
}
l.passband = 32;
l.stopband = 36;
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 803.1;
config_lpf(&l);
if (l.error) {
 /* report */
}
l.passband = 32;
l.stopband = 36;
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 803.6;
config_lpf(&l);
if (l.error) {
 /* report */
}
l.passband = 32;
l.stopband = 36;
```

```
l.passband = 32;
l.stopband = 36;
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 802.1;
config_lpf(300,&l);
if (l.error) {
 /* report */
}
l.passband = 32;
l.stopband = 36;
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 802.6;
config_lpf(&l);
if (l.error) {
 /* report */
}
l.passband = 32;
l.stopband = 36;
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 803.1;
config_lpf(&l);
if (l.error) {
 /* report */
}
l.passband = 32;
l.stopband = 36;
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 803.6;
config_lpf(&l);
if (l.error) {
 /* report */
}
l.passband = 32;
l.stopband = 36;
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 804.1;
config_lpf(&l);
if (l.error) {
 /* report */
}
```

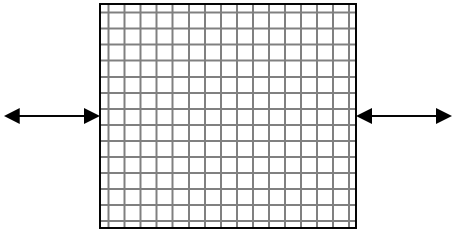
```
if (l.error) {
 /* report */
}
l.passband = 32;
l.stopband = 36;
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 802.1;
config_lpf(300,&l);
if (l.error) {
 /* report */
}
l.passband = 32;
l.stopband = 36;
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 802.6;
config_lpf(&l);
if (l.error) {
 /* report */
}
l.passband = 32;
l.stopband = 36;
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 803.1;
config_lpf(&l);
if (l.error) {
 /* report */
}
l.passband = 32;
l.stopband = 36;
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 803.6;
config_lpf(&l);
if (l.error) {
 /* report */
}
l.passband = 32;
l.stopband = 36;
l.passtol = 0.2;
l.stoptol = 0.1;
l.cfreq = 804.1;
config_lpf(&l);
```

(code shown is notional)

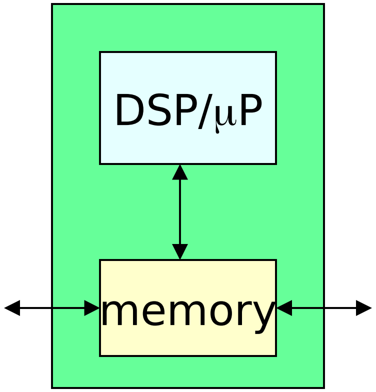
Different types of SPS hardware



hardware modem
fixed pipeline



FPGA
waveforms must share circui



Processor
operations performed by instructions

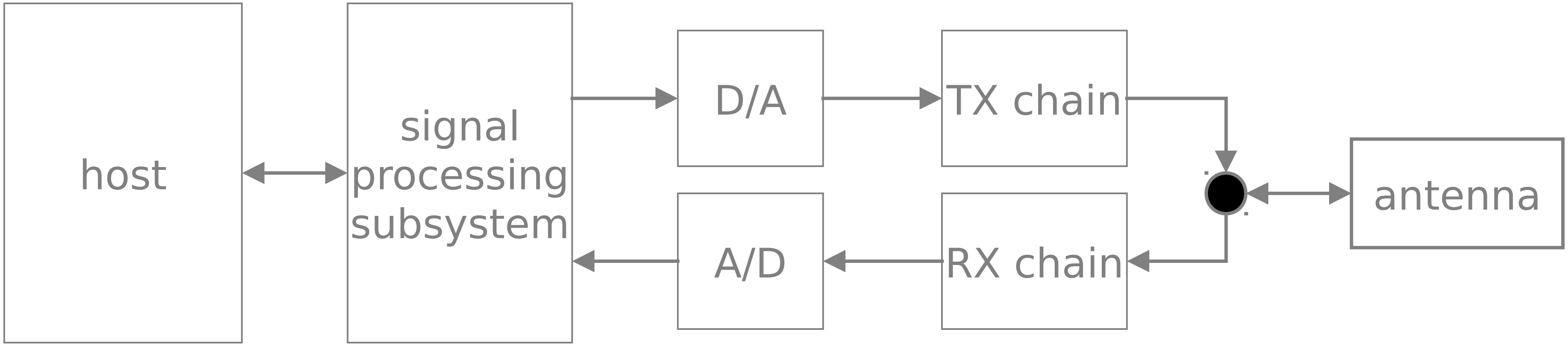
Configuring different kinds of SPS hardware

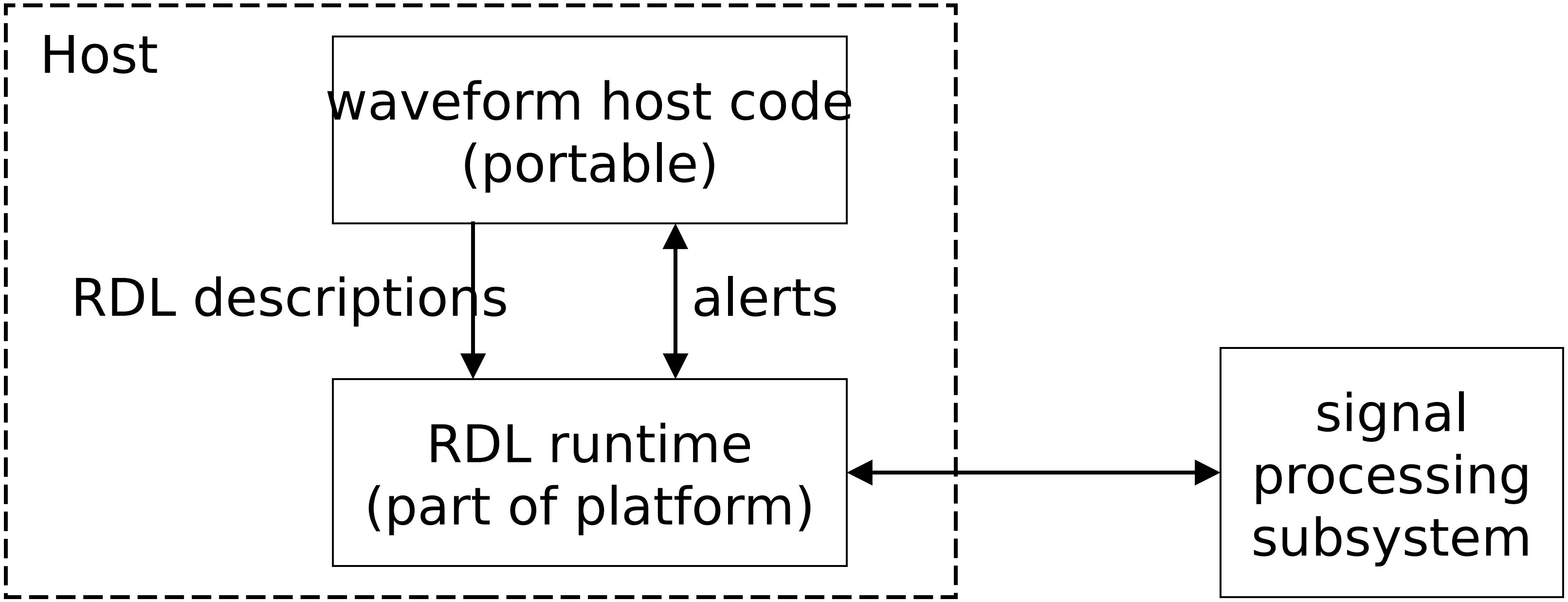
SPS type	on startup	on mode change
Hardware modem	no action	set parameters
FPGA	select map download map	set parameters rewire map
Processor	create objects connect objects	set parameters rewire objects create/delete objects

- Different operations required for different SPS types
- No matter how good the modem API is:
  - significant host code changes during porting
  - creates high porting costs

System architecture

RDL role in SDR system




Host
waveform host code (portable)
RDL descriptions
alerts
RDL runtime (part of platform)
signal processing subsystem

RDL concepts: Description

- Description
  - an RDL file that says what signal processing work to do

```
assembly FmRadio
{
  module IntegratedRfSource  src;
  module RfController        ctrl;
  module FmDemod             fm;
  module LowPassFilter       lpf;
  module OssDspSink          spkr;

  src -> fm -> lpf -> spkr;

  ctrl.ctrlout -> src.ctrlin;
}
```

RDL concepts: RDL runtime

- RDL runtime
  - the software components that link the host code to the SPS

- functions
  - configure the SPS as required by RDL descriptions
  - control and monitor the SPS during operation
  - communicate to/from the host code using alerts
    - like an API, except bidirectional